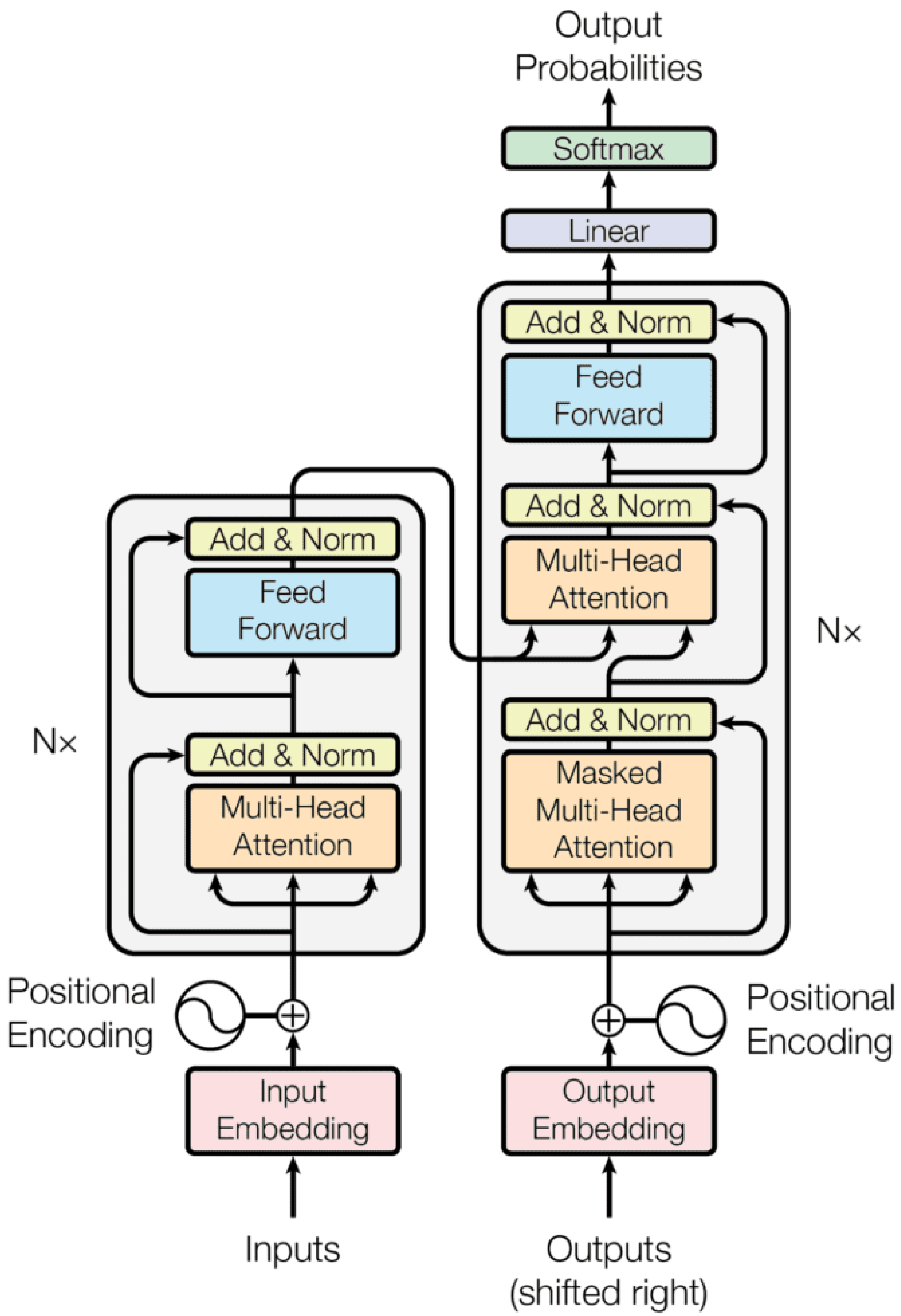
Transformers are a neural network with a novel architecture that aims to solve sequence-to-sequence complex language tasks like translation, question answering, and chatbots, all while managing long-range dependencies.

In cases where the data input is in textual form, a transformer model works by tracking relationships between the different words found in a given sentence. Moreover, transformers offer much longer memory withholding, handling long-range dependencies (more on that later in the article).

Transformers Architecture


Output Probabilities
Softmax
Linear
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Add & Norm
Masked Multi-Head Attention
Nx
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Nx
Positional Encoding
Positional Encoding
Input Embedding
Output Embedding
Inputs
Outputs (shifted right)

In the first iteration, the model has no context. It is clueless, it is going in random direction and trying to find the best way to converge. So that is what ML and DL is actually all about, is having all these parameters in the **adding and normalizing, feed forward network , even multi-head attention**.
We are trying to optimize the parameters for producing an output.

How Transformer model works?

- **Positional Encoding**
  First, we send a bunch of inputs into a transformer model and they passed in the positional encoding.
  In the Transformers network architecture, positional encoding is used to provide the model with information about the relative positions of the words in the input sequence. This is important because the Transformer model itself doesn't have any built-in notion of word order.

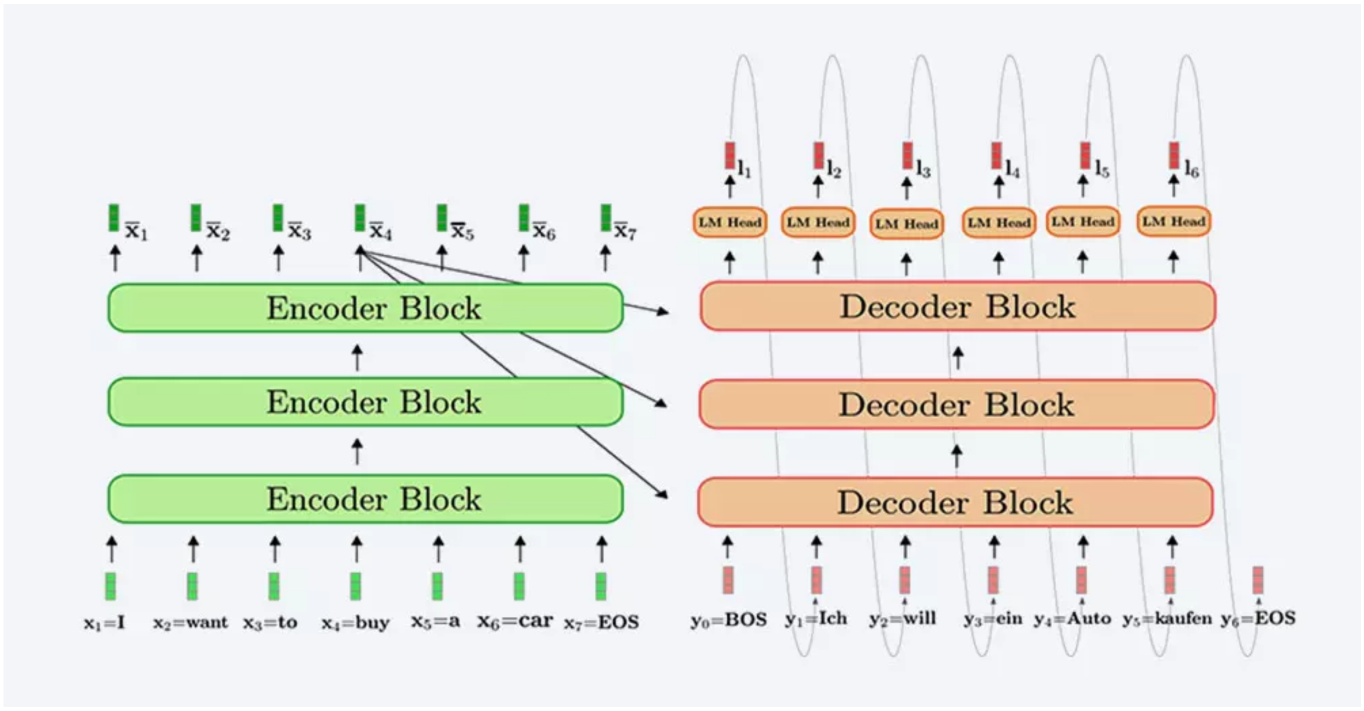
The reason positional encoding is necessary is because the Transformer model relies solely on self-attention mechanisms to capture dependencies between words in a sequence. Self-attention allows the model to weigh the importance of different words within the context of the entire sequence, but it doesn't inherently capture positional information.

Without positional encoding, the model would treat words as unordered sets, which would lead to a loss of important sequential information. For example, in the sentence "I love cats," the meaning changes if we swap the positions of the words: "Cats love I." The positional encoding helps the model distinguish between these different word orders.

The positional encoding is typically added to the input embeddings of the Transformer model. It is a fixed-length vector that is added element-wise to the word embeddings. The values in the positional encoding are carefully designed to encode information about the position of each word in the sequence.

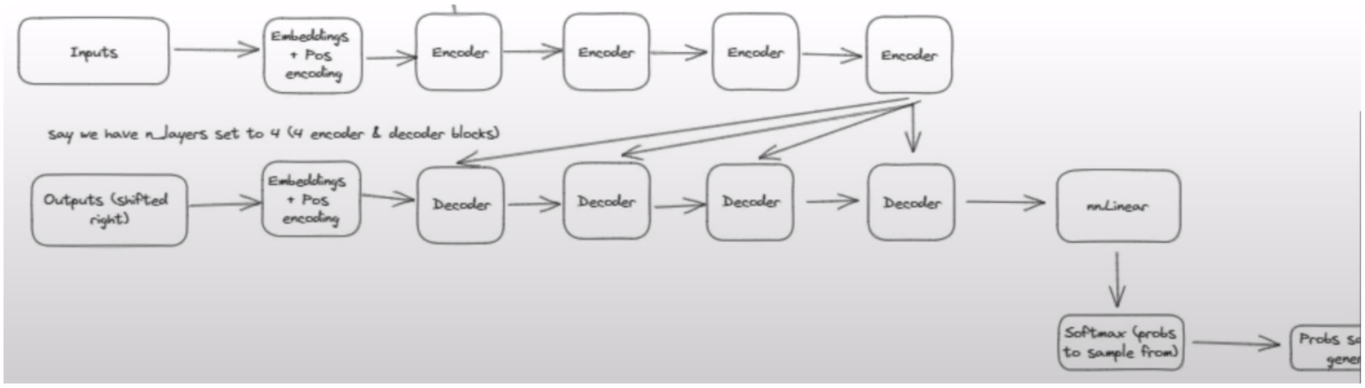
The most commonly used positional encoding is based on sine and cosine functions. Each dimension of the positional encoding corresponds to a specific frequency of the sine and cosine functions, allowing the model to capture different positional patterns. The values of the positional encoding are learned during training along with the other model parameters.

- **Encoders and decoders**


Encoder Block
Encoder Block
Encoder Block
Decoder Block
Decoder Block
Decoder Block
LM Head
$\bar{x}_1$ $\bar{x}_2$ $\bar{x}_3$ $\bar{x}_4$ $\bar{x}_5$ $\bar{x}_6$ $\bar{x}_7$
l_1 l_2 l_3 l_4 l_5 l_6
x_1=I x_2=want x_3=to x_4=buy x_5=a x_6=car x_7=EOS
y_0=BOS y_1=Ich y_2=will y_3=ein y_4=Auto y_5=kaufen y_6=EOS

A Transformer's decoder is made up of layers mounted one after the other, but each taking the output of the last encoder as an additional input. Thanks to this system, in each step of the decoder, we look to see if important information is in the past, i.e., in the encoder. By analogy, when translating a sentence, this would mean rereading the sentence to be translated at each word/step.

This is a more detailed image for how Transformer model work generally. Let's explain it!!

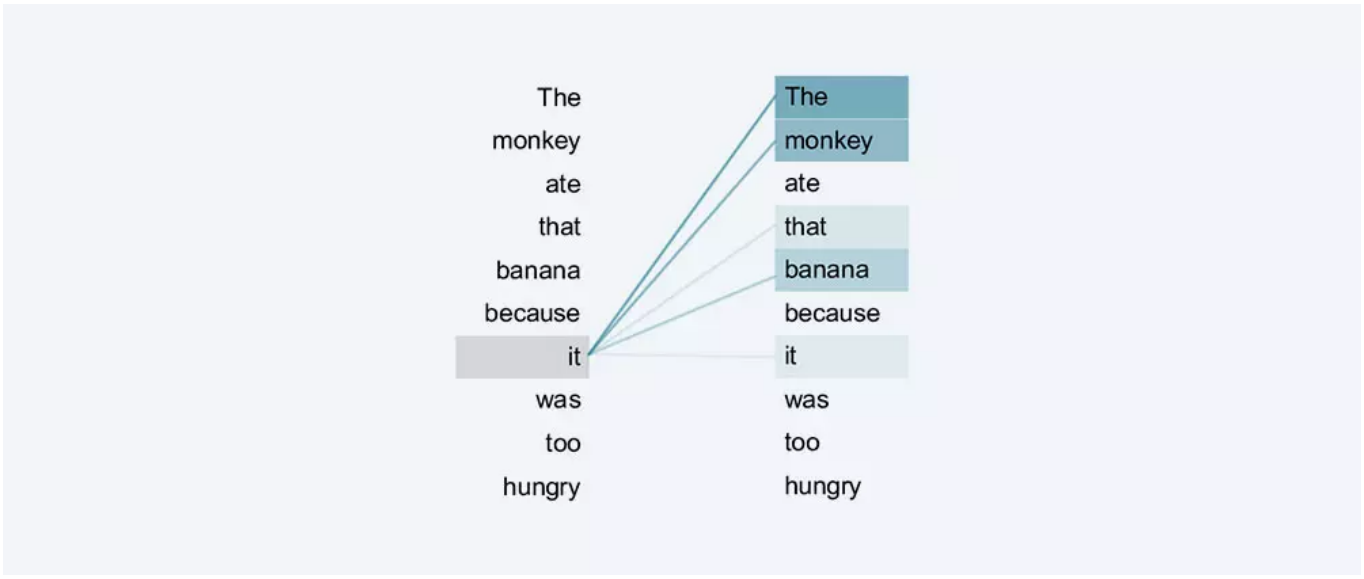

Inputs
Embeddings + Pos encoding
Encoder
Encoder
Encoder
Encoder
say we have n_layers set to 4 (4 encoder & decoder blocks)
Outputs (shifted right)
Embeddings + Pos encoding
Decoder
Decoder
Decoder
Decoder
nn.Linear
Softmax (probs to sample from)

We have some inputs and some outputs which are give to the model. We give each of them some ***embedding and positional encoding***. After that, we feed that in the first encoder layer(***we have n layers set to 4. If we have 4 encoders, we have 4 decoders***) and the next as soon as we hit the last one, we feed these into each of these decoders(***only the last encoder will feed on these decoders***).
These decoders will learn several things and then turn what the learn to the last decoder. They will apply a **linear transformation** at the end of the last decoder to simplify or give a summary of what it learned. Then we apply **softmax** on that new tensor to get some probabilities to sample. Once we get these probabilities, we can then sample from them and generate token.

- **Attention mechanism**

The idea of the Transformer is to preserve the interdependence of the words (tokens) of a sequence by using the attention mechanism which is at the center of its architecture. **This concept of attention measures the link between two elements of two sequences.** Thus in NLP, the attention mechanism makes it possible to transmit information to the model, so that it focuses its attention on the right place on the words of sequence A, when we process a word of sequence B.

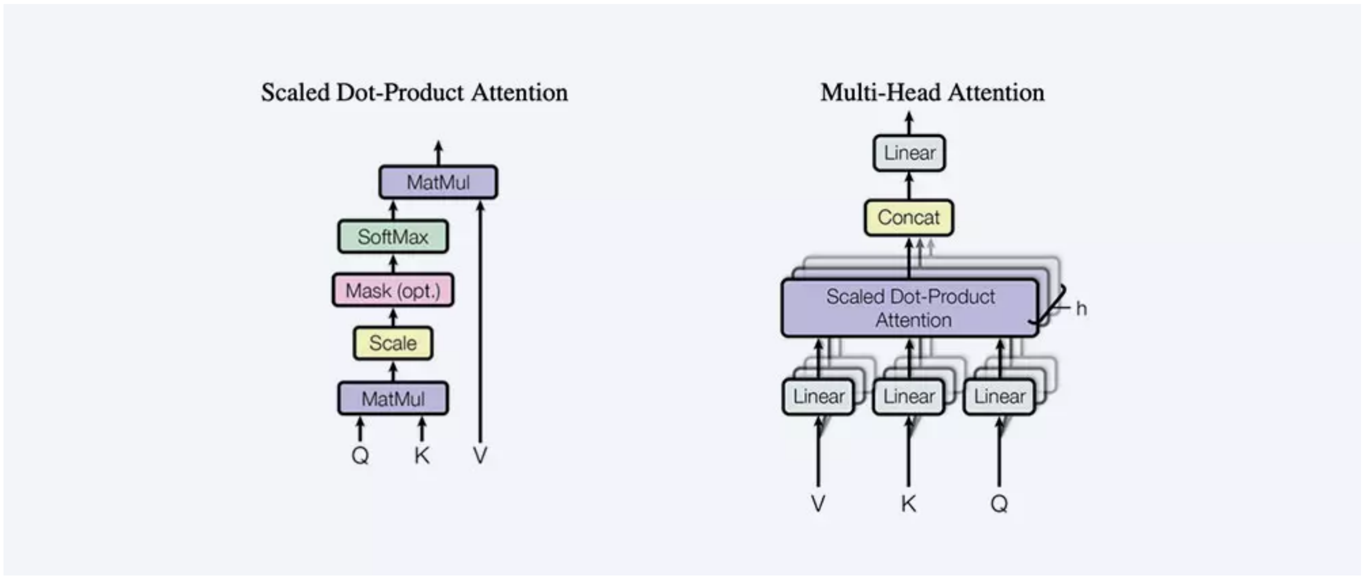
Self-attention is the attention mechanism but applied to a single sequence. The figure below shows that for the word “it”, the attention coefficient is high for “The monkey”. Which means that the model will have to rely on “The monkey” to encode “it”.



- Q is the Query vector
- K is the Key vector
- V is the Value vector
- dk is the dimension of the Key vectors

The calculation of this attention coefficient is done using the formula above where the three vectors are multiplied by the embedding of the input sequence. To popularize, in a search engine, Q would be the search request, K would be the characteristics (texts, images, etc.) associated with the best suited results V. The attention weights are divided by��dkto stabilize the gradients during training, then pass through a softmax function which normalizes the weights in order to select the Values to keep. These steps, called “attention heads,” are illustrated in the figure below.

The advantages of **multi-head attention** are to provide diversification of results when using several self-attention/heads of attention ($h$htimes), because the vectors Q, K and V are initialized randomly. The sequence of these attention heads makes learning more robust and parallelizable.



Source

https://www.openstudio.fr/2022/02/08/fonctionnement-des-transformers-et-leurs-applications/